Eurographics Symposium on Parallel Graphics and Visualization (2014)
M. Amor and M. Hadwiger (Editors)

Collaborative high-fidelity rendering over peer-to-peer networks

K. Bugeja[1] and K. Debattista[1] and S. Spina[1] and A. Chalmers[1]

[1]WMG, University of Warwick, Coventry, UK

Abstract

Due to the computational expense of high-fidelity graphics, parallel and distributed systems have frequently been employed to achieve faster rendering times. The form of distributed computing used, with a few exceptions such as the use of GRID computing, is limited to dedicated clusters available to medium to large organisations. Recently, a number of applications have made use of shared resources in order to alleviate costs of computation. Peer-to-peer computing has arisen as one of the major models for off-loading costs from a centralised computational entity to benefit a number of peers participating in a common activity. This work introduces a peer-to-peer collaborative environment for improving rendering performance for a number of peers where the program state, that is the result of some computation among the participants, is shared. A peer that computes part of this state shares it with the others via a propagation mechanism based on epidemiology. In order to demonstrate this approach, the traditional Irradiance Cache algorithm is extended to account for sharing over a network within the presented collaborative framework introduced. Results, which show an overall speedup with little overheads, are presented for scenes in which a number of peers navigate shared virtual environments.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.1]: Hardware Architecture—Parallel processing; Computer Graphics [I.3.2]: Graphics Systems—Distributed/network graphics; Computer Graphics [I.3.7]: Three-Dimensional Graphics and Realism—Animation and Raytracing; Computer Graphics [I.3.8]: Applications—; Systems and Software [H.3.4]: Distributed systems—

1. Introduction

Interactive high-fidelity rendering is increasingly being used in fields such as engineering, architecture, archaeology and defence. The physically-based approach behind such visualisation fidelity is computationally expensive due to the Monte Carlo methods typically used in such rendering solutions. The algorithms employed in high-fidelity rendering benefit greatly from added computational resources when exploiting parallelism, usually in the form of parallel shared memory architectures, distributed systems, vector processors or some hybridisation thereof. Many parallel rendering algorithms have been targeted at dedicated resources which are available to medium to large organisations. In contrast to traditional client-server architectures, peer-to-peer (P2P) computing has arisen as one of the major models for off-loading costs from a centralised computational entity to benefit a number of peers participating in a common activity. While removing the need for a central authority, the P2P model provides advantages in terms of scalability, burden-sharing and fault-tolerance. In the context of interactive rendering and visualisation, peers interacting within some environment will invariably compute and visualise similar portions of that environment. If the result of such computations is marshalled in a global state shared across participating peers, then the peers may be provided with results to computations they haven't yet carried out but may need to in the future. Reducing redundant computation by means of a global shared state introduces a number of challenges. Specifically, it must be ensured that (i) any changes to global shared state originating at a peer will eventually propagate through the network and, (ii) cumulative operations on global shared state are correctly sequenced in order to provide some agreed level of consistency.

In this paper we introduce the concept of collaborative high-fidelity rendering over P2P networks with the aim of furthering the quality of the rendering by reducing redundant computation. The irradiance cache (IC) algorithm [WRC88] is used in a collaborative environment as a case study to test

DOI: 10.2312/pgv.20141079

the viability and validity of the proposed system. Thus, our main contributions are:

- an event-based encapsulation of changes to global state
- an epidemiologic method for event propagation
- novel collaboration algorithm for high-fidelity rendering
- collaboration case study using the IC algorithm

2. Related Work

A detailed evaluation of parallel rendering techniques for high-fidelity rendering is given by Chalmers et al. [CDR02]. There have been many attempts at parallelising high-fidelity rendering; Muuss et al. [Muu95] presented an architecture for interactive ray tracing on 96 processors. Parker et al. [PMS*99] presented an interactive ray tracer that used frameless rendering on a dedicated 64-core shared memory multiprocessor; it was then adapted by DeMarle et al. [DPH*03] for cluster systems. In particular, the system was enhanced to provide an object-based shared memory implementation over distributed memory via demand paging. While these approaches used expensive supercomputers, Wald et al. [WSBW01, WBS02] introduced a parallel ray tracer running on a distributed system which achieved interactivity through careful use of instruction level parallelism via SIMD instructions. This work was later extended to provide a full global illumination solution in [WKB*02], achieving quasi-linear speed up on 48 processors.

2.1. Large Scale Distributed Rendering

Distributed approaches to high-fidelity rendering include the use of GRID computing, with algorithms adapted to shared resources. Aggarwal et al. [ACD08] presented a two-stage rendering system for computational grids based on the irradiance cache and, in follow up work [ADD*09], introduced rendering on desktop grids wherein single images were computed within user-based time constraints. This work was finally extended to interactive high-fidelity rendering [ADBR*12]. Other distributed approaches include BURP (Berkeley Ugly Rendering Project) [PGAB*09], which is based on the Boinc framework [And04] and makes use of volunteer computing to perform large-scale rendering. Mateos et al. [RGMFLL09] introduced Yafrid-NG, a physically-based renderer which makes use of a P2P architecture to speed up rendering by distributing computation over the internet to a set of heterogeneous machines. These solutions are nonetheless tailored towards offline rendering, as opposed to interactivity that this system aims to achieve in the future. Crucially, all rendering approaches mentioned make use of a master-slave paradigm to distribute computation to worker nodes, notwithstanding the degree of decentralisation employed.

2.2. Peer-to-peer Systems

P2P architectures have been used in data sharing, collaboration and for information dissemination. The decentralised nature of these systems addresses scalability problems in distributed applications that exist when the number of clients starts to grow. P2P approaches aimed at sharing resources and information require efficient search mechanisms to locate required information in a timely manner. In local-area solutions, unstructured systems use multicasting facilities provided by the underlying hardware to broadcast queries for specific data. In large scale networks, implementing reliable multicasting is notoriously difficult [JvS02]. An approach adopted by unstructured P2P systems was that of query flooding, whereby all reachable nodes are contacted to determine the availability of a resource on the network. Structured P2P systems such as Chord [SMK*01] and Tapestry [ZKJ*01] avoid the traffic caused by query flooding by adopting key-based routing and searching. Specifically, a distributed hash table system is used to provide a lookup service similar to an associative array; the search space is partitioned and the search criteria are associated with hosts holding the required resources.

A series of randomised algorithms for replicated database maintenance based on epidemic principles was introduced by Demers et al. [DGH*87]. This addressed problems of high traffic and database inconsistency, and was later exploited in Bayou [DPS*94], a system providing support for data sharing and collaboration among weakly connected users, which used peer-to-peer anti-entropy for the propagation of updates. Jelasity et al. [JvS02] conceived the newscast model of computation, providing effective and reliable probabilistic multicasting, large-scale distributed file-sharing, and resource discovery and allocation, with the distinguishing feature being the membership protocol employed. A peer may contact any arbitrarily chosen member and simply copy that member's list of neighbours in order to join a group. Leaving a group is achieved by that peer merely ceasing its communication as opposed to notifying other members in the group about its decision.

Collaborative peer-to-peer high-fidelity rendering has never been suggested before, despite its potential in providing a scalable, fault-tolerant platform for sharing rendering computation. This work builds upon the membership and update propagation principles above [JvS02, DGH*87] to provide a framework for collaborative high-fidelity rendering via the use of event propagation for the maintenance of a global shared state across peers.

3. Method

This section provides an overview of our method. The assumption here is that during image synthesis, each individual peer may need to carry out some computation that has already been effected by another peer, thus laying the groundwork for potential collaboration. Such computations usually entail populating data structures for caching, interpolating or generally speeding up the computation of indirect lighting. In a collaborative context, these data structures, along with others which may hold important state information describing the active scene, become part of a larger state that is shared across all the participants, P. Let $S_i = L_i \cup G_i$ be the state of participant $P_i \in P$, where L_i is P_i's internal state and G_i is the shared state.

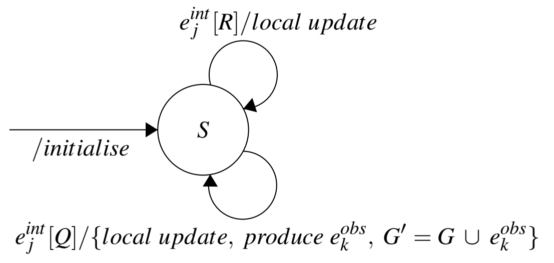
$e_j^{int}[R]/local\ update$
$/initialise$
S
$e_j^{int}[Q]/\{local\ update,\ produce\ e_k^{obs},\ G' = G \cup e_k^{obs}\}$

Figure 1: *Internal event loop and observable event generation.*

3.1. Observable Events

An internal event e^{int} is the result of a write that modifies internal state L_i and potentially influences shared state. An observable event e^{obs} is the result of any significant change in internal state and is responsible for exposing a series of events $e_n^{int}, e_{n+1}^{int}, \ldots, e_{m-1}^{int}, e_m^{int}$ as part of the shared state G_i. In practice, an observable event is abstracted to delineate and encapsulate a number of smaller, logically-related changes. For example, in an algorithm such as the irradiance cache, where irradiance samples are computed and inserted into an octree, a significant change would be represented by the insertion of an agglomeration of generated samples into the acceleration structure, as opposed to that of a single sample. An enumeration of P gives an index set $I \subset \mathbb{N}$, where $f : I \to P$ is the particular enumeration of the set of participants P. A useful abstraction we adopt is the grouping of all observable events into the global state G as represented by the shared state at each peer P_i, such that $G = \bigcup_{i \in I} G_i$. In line with our definition of an observable event as a significant change in state, we also define a mechanism by which we can determine what qualifies as a significant change in state. Specifically, for an event e_j^{int}, if the state S_i satisfies a predicate Q, then an observable event e_k^{obs} is produced and merged with the global state G_i (Fig. 1). The sequence of observable events $e^{obs} \in G_i$ generated by P_i is guaranteed to be ordered in time, i.e., $e_m^{obs} \to e_{m+1}^{obs}$, where $a \to b$ means event a precedes event b insofar as events are generated by the same peer. It would be desirable to extend this guarantee to events generated by other peers, establishing some form of event ordering across all collaborating peers, since observable events may represent cumulative or dependent updates to shared data structures. For example, in the case of events that invalidate the contents of a data structure, one should be able to establish whether a specific event has happened before, after or was concurrent to the invalidation.

3.2. Logical Order of Events

In distributed systems, especially in decentralised applications, time-of-day clocks may be skewed or suffer from drift, and thus, global (or absolute) time ordering of events could lead to unexpected results [Krz]. Instead, we choose to capture the causal order of events by employing logical clocks instead of physical ones [Lam78]. A logical clock is an n-tuple $n \leq |P|$; a participant P_i is responsible for incrementing the i^{th} element of its logical clock whenever an observable event occurs [Fid88]. With the help of a logical timestamp function $\mathcal{V}(e^{obs})$, we attempt to determine the system-wide ordering for an event e^{obs}; consider two events e_i^{obs} and e_j^{obs} with vector timestamps $\mathbf{V} = \mathcal{V}(e_i^{obs})$ and $\mathbf{V}' = \mathcal{V}(e_j^{obs})$ respectively:

Rule 1: e_i^{obs} happens before e_j^{obs} ($e_i^{obs} \to e_j^{obs}$) when each element of $\mathbf{V}$ is less or equal to the respective element in $\mathbf{V}'$, i.e., $e_i^{obs} \to e_j^{obs} \iff \mathbf{V}[n] \leq \mathbf{V}'[n]$ for $n \in I$ (see §3.1).

Rule 2: $\mathbf{V}$ and $\mathbf{V}'$ are said to be equal if their respective elements are equal, i.e. $\mathbf{V} = \mathbf{V}' \iff \mathbf{V}[n] = \mathbf{V}'[n]$ for $n \in I$.

Rule 3: Events e_i^{obs} and e_j^{obs}, with timestamps $\mathbf{V}$ and $\mathbf{V}'$ where $\exists\, n,m \in I : (\mathbf{V}[n] > \mathbf{V}'[n]) \wedge (\mathbf{V}[m] < \mathbf{V}'[m])$, are not causally related but rather denote concurrent events.

A peer generates observable events and tags them with a logical timestamp. The logical clock of the peer is also updated to reflect the generated event (Alg. 2). Events are then communicated to other peers via a propagation mechanism (see §3.3). The three rules above are used to determine the order of the propagated events before they are committed at the receiving peer. In case of conflict due to difficulty ordering concurrent events, a tie breaking function $T(e_i^{obs}, e_j^{obs})$ is used to deterministically resolve the tie in favour of one event or the other. Each observable event generated may increase the cardinality of the global state G (since $G = \bigcup_{i \in I} G_i$), and in scenarios where event generation occurs at high frequencies, this can result in uncontrolled growth. We mitigate this growth via the use of special observable events called *grouping events* (e^{grp}), and use them to group a set of observable events into a single event, retiring the members of the set in the process.

3.3. Event Propagation

The propagation of observable events between peers employs strategies common to epidemic processes [B*75]. Specifically, an *anti-entropy* strategy is used whereby each peer regularly chooses another peer at random (see §3.4) with which to exchange observable events (Alg. 1). The aim of this exchange is that of harmonising the global state of each peer such that for peers P_i and P_j, $G_i = G_j$. A peer that has not yet seen an observable event is *susceptible* to it, while a peer that has generated or can provide the event is called *infective*. A peer that assimilates an observable event into a grouping event is known as *retired* with respect to that event. This terminology is loosely based on Kermack et al. [KM32]. The dynamics of event propagation are based on the *push* dynamics employed in epidemic models [DGH*87]. Thus, propagation may be modelled using established deterministic techniques from epidemiology literature. When an observable event is generated at a peer, propagation through the network is achieved in expected time proportional to the log of the number of peers, $n = |P|$. Specifically, for large n, the exact formula is $\log_2(n) + \ln(n) + O(1)$ [Pit87]. The exchange of events is a two-step process, whereby the global state of each peer is merged with the other's, if found diverging (see Alg. 3). In particular, consider two peers P_s and P_d, where P_s is the originator of the exchange and P_d the recipient. If their respective shared state, G_s and G_d, are found differing, G_s is merged with G_d at the recipient, while G_d is merged with G_s at the originator, in that order. During each merge process, the events from both states are first combined and ordered. The newly acquired events are then committed in their perceived order of occurrence. During event exchange, the logical clocks of the respective peers are updated and made consistent (Alg. 2). An example of event propagation, ordering and merging is shown in Figure 2. In this example, four peers (P_1 through P_4) participate in the network. Three observable events (a, b, c) are generated by P_1, P_2 and P_4 respectively. The resulting timestamps can be observed below the events for the respective peers. Subsequently, an exchange ensues between P_1 and P_2, and another between P_3 and P_4. In the first case we see that events a and b are concurrent (**Rule 3**) and thus resort to a tie-breaking function which orders events by process id, before merging and resulting in $G_1 = G_2 = \{ab\}$. In the second exchange, between P_3 and P_4, P_3 simply acquires the event, such that $G_3 = G_4 = \{c\}$. Next, P_1 leaves the network, P_2 and P_3 start an exchange, while P_4 produces an invalidation event d. The exchange between P_2 and P_3 results in $G_2 = G_3 = \{abc\}$, while the invalidation event d at P_4 results in c being removed. In the next stage, P_2 leaves the network and a final exchange ensues between P_3 and P_4. The ordering process promotes d as the most recent event by virtue of it being an invalidation event; the tie-breaking function between an invalidation event and a standard event will always break the tie in favour of the standard event (i.e., we assume the standard event happened before the invalidation event). Concretely, a standard event may, for example, be thought of as the agglomeration of a number of irradiance samples computed and inserted into an irradiance cache. The invalidation event could be any event that invalidates these samples, such as the flicking of a light switch, triggering a light source on and off which was used in generating these samples (see §4).

Algorithm 1 Anti-entropy Propagation Algorithm

```
Require: $|P| \geq 1 \wedge \exists\, p \in P : active(p)$
1: while $active(P_s)$ do
2:     $P_d \Leftarrow$ ChoosePeer($P$)
3:     ExchangePeers($P_s, P_d$)                ▷ see §3.4
4:     ExchangeEvents($P_s, P_d$)
5:     Sleep($\Delta t$)                ▷ delay $\Delta t$ before next exchange
6: end while
```

Algorithm 2 Updating of logical clocks

```
1: function UpdateClock($P_s$, $P_d$)
2:     Tick($P_s$)
3:     $\mathbb{V}'_{P_d} \Leftarrow \sup(\mathbb{V}_{P_s}, \mathbb{V}_{P_d})$
4:     Tick($P_d$)
5: end function
6:
7: function Tick($P_i$)
8:     Increment $\mathbb{V}_{P_i}[i]$ by 1
9: end function
```

Algorithm 3 Global state synchronisation algorithm

```
1: function ExchangeEvents($P_1$, $P_2$)
2:     if $G_1 \neq G_2$ then
3:         UpdateClock($P_1, P_2$)
4:         MergeEvents($G_1, G_2$)
5:         UpdateClock($P_2, P_1$)
6:         MergeEvents($G_2, G_1$)
7:     end if
8: end function
9:
10: function MergeEvents($G_s$, $G_d$)
11:     $C \Leftarrow G_s \cap (G_s \setminus G_d)$
12:     Commit($C, R$) where $R = \{x, y \in C : Pre(x, y)\}$
13: end function
14:
15: function Pre($e_1$,$e_2$)
16:     if $e_1 = e_2$ then
17:         return $false$
18:     end if
19:     if $\mathcal{V}(e_1) = \mathcal{V}(e_2)$ then
20:         return $T(e_1, e_2)$
21:     else if $\mathcal{V}(e_1) \leq \mathcal{V}(e_2)$ then
22:         return $true$
23:     else if $\mathcal{V}(e_2) \leq \mathcal{V}(e_1)$ then
24:         return $true$
25:     else
26:         return $T(e_1, e_2)$
27:     end if
28: end function
```

3.4. Peer Discovery and Membership

Unstructured P2P systems aim at exploiting randomness to disseminate information across a large set of

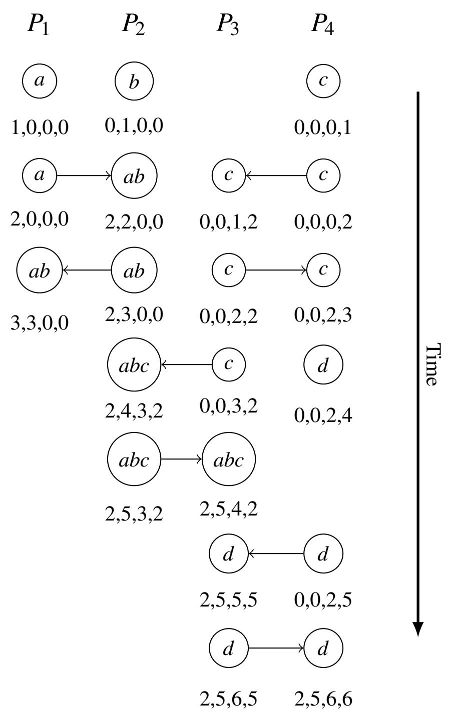

Figure 2: *Example of event propagation, ordering and merging of global states.*

nodes [VGVS05]. Thus, membership, or the way a collaborating peer learns about other peers, is fundamental because it controls the performance of subsequent disseminations. Connections between nodes in gossiping networks are highly dynamic and often need to obtain random samples from the entire network in order to periodically exchange information with random peers. Membership is handled in a number of different ways; a straight forward approach is that of furnishing the peers with a fixed directory provider, which lists all collaborating peers in the network. This approach requires maintaining global information which may prove to be problematic, especially in the case of major network disasters [VGVS05]. The approach we take is similar to the newscast method [JvS02], where we opt to keep a finite cache of peers instead of all the members of the network. Each peer is tagged with a logical timestamp representing the last communication event associated with it. The cache needs to be cold started by populating it with at least one peer which is already a member of the network. Subsequently, at each exchange caused by the anti-entropy algorithm (Alg. 1), the caches of the two peers taking part in the exchange are merged, possibly resulting in a number of peers twice the size of each individual cache. The eviction policy used is conceptually similar to a *least recently used* strategy, where the list of peers is sorted by their logical timestamp and the top k entries are retained, where k is the size of an individual cache.

4. Case Study: Irradiance Caching over P2P

The irradiance cache is an object-space algorithm; it computes and caches diffuse interreflection values which are independent of any observer (view-independent). Therefore, for multiple observers, irradiance values computed by one may be reused by another without any need for recomputation, making the technique a good case study for our system. Multiple peers interacting within the same scene may contribute to the generation and sharing of samples in the irradiance cache, making the data structure a shared one among collaborators. When objects in the scene move or the parameters of light sources change, the affected samples in the cache become stale. In such cases, we take a straightforward approach and invalidate the whole irradiance cache. To capture this behaviour, we define two kinds of observable events, one for adding samples to the irradiance cache, E_{ins}, and another for invalidating them, E_{clr}, fundamentally clearing the structure. These events are captured by the predicates:

$$Q_{ic} = size_{curr} - size_{prev} \geq size_{epoch} \tag{1}$$

$$\begin{aligned} Q_{chg} = \exists\, x \in objects : hasMoved(x) \vee \\ \exists\, y \in lights : hasChanged(y) \end{aligned} \tag{2}$$

where $size_{curr}$ is the number of generated irradiance samples in the cache, $size_{prev}$ is the number of generated irradiance samples at time $t - \Delta t$, and $size_{epoch}$ is the number of generated samples required to trigger an observable event. With respect to the changes captured by `hasMoved` and `hasChanged`, we keep track of orientation and position changes in objects and intensity of light sources. For every observable event, a Universally Unique Identifier (UUID) is generated to differentiate it from every other observable event in the system. The event is also timestamped using a vector clock to help establishing a system-wide ordering.

4.1. Event Merging

During an exchange between two peers, observable events are ordered and merged (see Alg. 1, 3). The merging mechanism for two events e_n^{obs} and e_m^{obs} of type E_{ins} is straightforward; the insertion of samples into the irradiance cache is order independent, unless another event of type E_{clr} occurred between e_n^{obs} and e_m^{obs}. During ordering, when invalidation events are concurrent to insertion events, the latter are always assumed to have happened before: $T(E_{ins}, E_{clr})$ always returns true, while $T(E_{clr}, E_{ins})$ always returns false if the events are concurrent. When the event list has been ordered, it is traversed from most to least recent, with each observable event of type E_{ins} resulting in an insertion of a

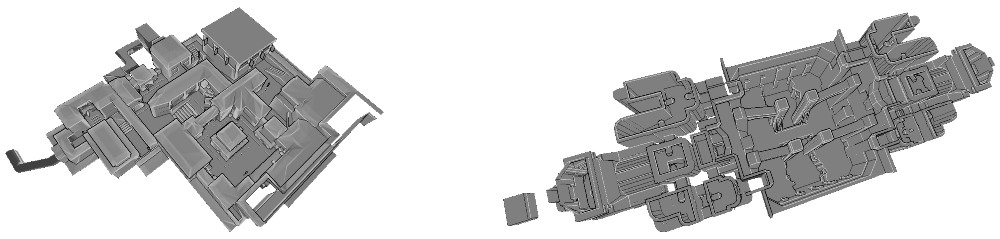

Figure 3: *Scenes used for experiments : Halflife community map (left) and Quake 3 Team Arena map (right).*

batch of samples into the irradiance cache. Each batch insertion is assigned an epoch number, which, on a given peer, uniquely identifies samples inserted as a result of a specific observable event. If an invalidation event is encountered during traversal, the irradiance cache is cleared from samples belonging to earlier epochs, and the traversal terminates before all events have been processed.

4.2. Event Grouping

The more dissemination cycles pass from the generation of an event, the less likely it is to find a peer that has yet to learn about the event. Therefore, during peer exchanges, longer lived events are less likely to contribute to the exchange, making their broadcasting, as time goes by, redundant. Event grouping is meant to aggregate such events to make exchanges more efficient. The premise of event grouping is based on rumour spreading, which is also founded in epidemic theory. In particular, when two peers try to exchange an event both have, the event has a probability $\frac{1}{k}$ of being merged into a group. This reflects the idea of a person (infective) who tries to spread a rumour within a group of n other persons (susceptible) by randomly calling people in the group, one at a time. When a call results in the other person already knowing the rumour, the caller starts losing interest in actively spreading the rumour [DGH*87]. Event grouping is aimed at reducing the amount of information exchanged by peers. When the intersection of two event groups is not the empty set, then the groups are merged together and a new group is created. Both peers share the same newly created group, retiring the two subsets.

5. Results

The method introduced in §3 has been evaluated using the case study presented in §4, implemented on top of Illumina [Bug14], an open-source physically-based rendering solution. Tests were run on a local network formed by connecting eight heterogeneous machines, each equipped with 8GB of memory and at quad-core chip multiprocessing capabilities. The datasets used for testing were purposefully borrowed from multi-player first-person videogames (Fig. 3). The use of such datasets coupled with a heterogeneous mixture of machines aim to improve the ecological validity of the study. For each of the two datasets considered, interest points were drawn up and from these a number of paths of variable length were randomly generated. A path does not necessarily span all the points of interest. Each machine was deterministically seeded with a path, which was subsequently used across all experiments carried out. Experiments were conducted for both individual and overall network speed up, under best, average and worst-case conditions. For the irradiance cache, an error value (α) of 0.15 was used and a total of 1.5k rays were traced for each computed sample. Each machine in the network runs a single peer, with four threads devoted to rendering. These peers are configured to attempt an exchange not earlier than $2.5s$ subsequent to the completion of a previous exchange. For a given peer, this means that the time elapsed between an exchange and the next is at least $2.5s$. A peer will query an incoming exchange request buffer every $250ms$. These settings were used for both scenes.

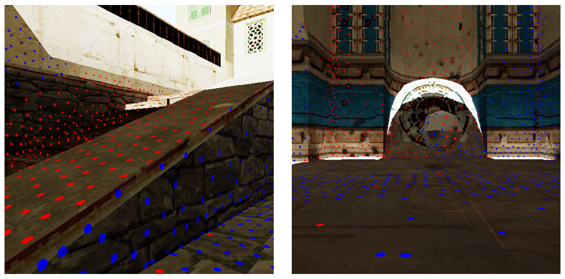

Figure 4: *IC sample distribution : red samples are computed locally, while blue samples are acquired over the network.*

5.1. Timing and Speed-up

In order to contextualise any possible gains due to collaboration, the rendering times for each peer and its selected path were recorded under worst and optimal conditions. Specifically, in the first instance, each peer was forced to perform a walkthrough of its path without any collaboration but with any overheads thereof. Following this, a second set of runs recorded the walkthrough rendering times for

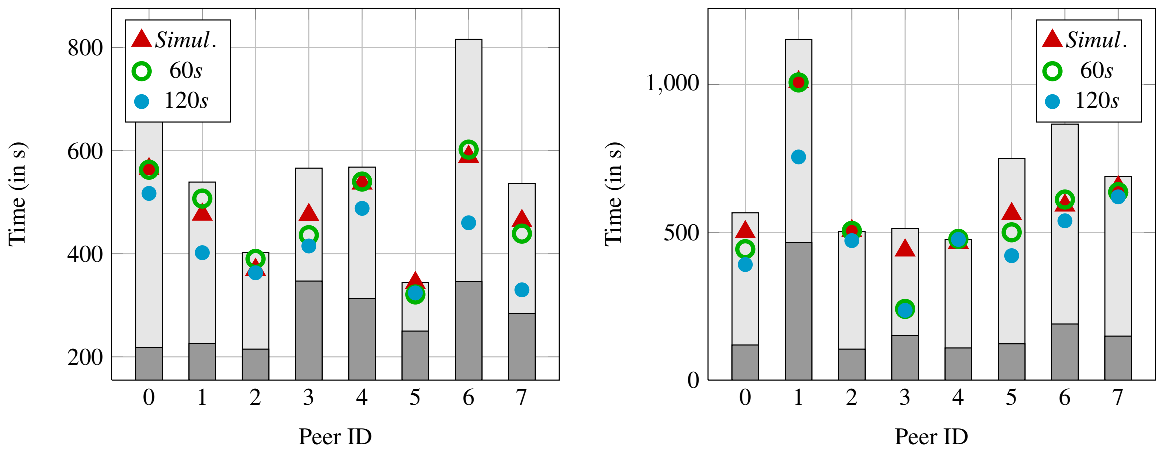

Figure 5: *Timings for Halflife community map (left) and Quake 3 Team Arena map (right) for (i) simultaneous startup, (ii-a) 60s staggered startup and (ii-b) 120s staggered startup. The grey bars show timings under best and worst cases.*

each peer when connected to a network that can provide an already fully computed irradiance cache. This reflects the best-case performance. Figure 5 illustrate this; light and dark grey bars show the worst and best-case performance respectively. Peers joining a network with a saturated irradiance cache demonstrate speed-ups between 2× and 5× with respect to those computing the irradiance cache themselves, for the same path. Multiple peers may concurrently generate samples covering the same areas; when an exchange ensues between these peers, samples added to the cache may cluster, slowing down irradiance interpolation. In order to avoid this, a minimum distance criterion is applied to samples acquired over the network, whereby a sample is rejected if another exists in the destination cache within some specified distance.

5.2. Simultaneous and Staggered Start

Figure 5 also illustrates the effect of the system on rendering times when (i) the network peers boot up simultaneously and (ii) peers join at intervals of 60s (ii-a) and 120s (ii-b). In (i), peers find an empty global IC on joining. The overall speed-up of the network is around 1.17× for both scenes; when the peers are considered individually, it can be observed that not all of them benefit from this speed-up. Newly generated samples may take time to propagate across the network, and in the meantime peers which could have benefitted from these samples would have computed their own (Fig. 4). Also, a peer may only take advantage of topical samples; even if it were to receive a substantial number of irradiance samples from another peer early on its walkthrough, these samples could only be used if the paths of the two peers intersect each other at some point. In (ii-a), although faster than the worst case (by 1.17× and 1.24× respectively), it did not differ much from (i), especially for *scene a*. In (ii-b), the average speed-up on the network showed improvement over (i) and the worst case (by 1.37× and 1.4× respectively).

6. Conclusion and Future Work

This paper presents a novel algorithm for high-fidelity collaborative rendering over P2P networks. The reference implementation and respective case study demonstrate that it is possible to take advantage of collaboration in P2P systems to speed up high-fidelity rendering. The novelty of this work lies in laying the foundation for P2P systems for high-fidelity rendering that provide an alternative to the status quo of centralised systems. However, a number of limitations remain that require further investigation and work to improve the potential of P2P rendering systems. A limitation of the system is that for propagation purposes, newly generated events are not given priority over older events. Subsequently, a recently generated event will not propagate until an anti-entropy operation is initiated (§3.3). This shortcoming has the effect of introducing some initial delay between the generation and the propagation of an event, which, in a frequently changing system, might be considered undesirable. Full advantage of event grouping could be taken (§3.2, §4.2), to prioritise propagation of recent events; this could be coupled with the use of resource mongering to push these events to neighbouring peers as soon as they are generated. Furthermore, no discrimination is made with respect to how specific events affect specific peers. To use the irradiance cache as an example, a given peer might be more interested in receiving events which pertain to samples in its vicinity, rather than events related to samples which are far away and may never be required. Thus, an investigation could be carried out to determine the feasibility of prioritising the delivery of events according to the needs of respective peers. This study was limited to 8 peers; it was observed that among the participants, weaker machines seemed to benefit the most, although this was not properly evaluated. Future work should investigate the scalability of the system and quantify any benefits participants reap depending on their specifications. Future work should also look into providing interactive rendering to the system. The rendering techniques currently employed are not based on ray packets and rendered using CPUs; this is computationally expensive and can be partially offloaded to GPUs with minimal effort via techniques such as splatting [GKBP05]. Minimising the rendering component when irradiance values can be trivially interpolated would highlight even more the benefits of collaboration.

References

[ACD08] Aggarwal V., Chalmers A., Debattista K.: High-fidelity rendering of animations on the grid: a case study. In *Proceedings of the 8th Eurographics conference on Parallel Graphics and Visualization* (2008), Eurographics Association, pp. 41–48. 2

[ADBR*12] Aggarwal V., Debattista K., Bashford-Rogers T., Dubla P., Chalmers A.: High-fidelity interactive rendering on desktop grids. *IEEE Computer Graphics and Applications 32*, 3 (2012), 24–36. 2

[ADD*09] Aggarwal V., Debattista K., Dubla P., Bashford-Rogers T., Chalmers A.: Time-constrained high-fidelity rendering on local desktop grids. In *Proceedings of the 9th Eurographics conference on Parallel Graphics and Visualization* (2009), Eurographics Association, pp. 103–110. 2

[And04] Anderson D. P.: Boinc: A system for public-resource computing and storage. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on* (2004), IEEE, pp. 4–10. 2

[B*75] Bailey N. T., et al.: *The mathematical theory of infectious diseases and its applications*. Charles Griffin & Company Ltd, 5a Crendon Street, High Wycombe, Bucks HP13 6LE., 1975. 4

[Bug14] Bugeja K.: Illumina prt: A parallel physically-based rendering system, 2014. 6

[CDR02] Chalmers A., Davis T., Reinhard E.: *Practical Parallel Rendering*. AK Peters Ltd, 2002. 2

[DGH*87] Demers A., Greene D., Hauser C., Irish W., Larson J., Shenker S., Sturgis H., Swinehart D., Terry D.: Epidemic algorithms for replicated database maintenance. In *Proceedings of the sixth annual ACM Symposium on Principles of distributed computing* (1987), ACM, pp. 1–12. 2, 4, 6

[DPH*03] DeMarle D. E., Parker S., Hartner M., Gribble C., Hansen C.: Distributed interactive ray tracing for large volume visualization. In *Proceedings of the 2003 IEEE Symposium on Parallel and Large-Data Visualization and Graphics* (2003), IEEE Computer Society, p. 12. 2

[DPS*94] Demers A., Petersen K., Spreitzer M., Ferry D., Theimer M., Welch B.: The bayou architecture: Support for data sharing among mobile users. In *Mobile Computing Systems and Applications, 1994. Proceedings., Workshop on* (1994), IEEE, pp. 2–7. 2

[Fid88] Fidge C. J.: Timestamps in message-passing systems that preserve the partial ordering. In *Proceedings of the 11th Australian Computer Science Conference* (1988), vol. 10, pp. 56–66. 3

[GKBP05] Gautron P., Křivánek J., Bouatouch K., Pattanaik S.: Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Proceedings of the Sixteenth Eurographics conference on Rendering Techniques* (2005), Eurographics Association, pp. 55–64. 8

[JvS02] Jelasity M., van Steen M.: Large-scale newscast computing on the internet. 2, 5

[KM32] Kermack W. O., McKendrick A. G.: Contributions to the mathematical theory of epidemics. ii. the problem of endemicity. *Proceedings of the Royal society of London. Series A 138*, 834 (1932), 55–83. 4

[Krz] Krzyzanowski P.: Lectures on distributed systems: Clock synchronization. *Lecture notes, Rutgers University Computer Science 416*. 3

[Lam78] Lamport L.: Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (1978), 558–565. 3

[Muu95] Muuss M. J.: To wards real-time ray-tracing of combinatorial solid geometric models. In *Proceedings of BRL-CAD symposium* (1995), Citeseer. 2

[PGAB*09] Patoli M., Gkion M., Al-Barakati A., Zhang W., Newbury P., White M.: An open source grid based render farm for blender 3d. In *Power Systems Conference and Exposition, 2009. PSCE'09. IEEE/PES* (2009), IEEE, pp. 1–6. 2

[Pit87] Pittel B.: On spreading a rumor. *SIAM Journal on Applied Mathematics 47*, 1 (1987), 213–223. 4

[PMS*99] Parker S., Martin W., Sloan P.-P. J., Shirley P., Smits B., Hansen C.: Interactive ray tracing. In *Interactive 3D Computer Graphics* (1999). 2

[RGMFLL09] Ramos J. M., González-Morcillo C., Fernández D. V., López-López L.: Yafrid-ng: A peer to peer architecture for physically based rendering. In *CEIG 09-Congreso Espanol de Informatica Grafica* (2009), The Eurographics Association, pp. 227–230. 2

[SMK*01] Stoica I., Morris R., Karger D., Kaashoek M. F., Balakrishnan H.: Chord: A scalable peer-to-peer lookup service for internet applications. In *ACM SIGCOMM Computer Communication Review* (2001), vol. 31, ACM, pp. 149–160. 2

[VGVS05] Voulgaris S., Gavidia D., Van Steen M.: Cyclon: Inexpensive membership management for unstructured p2p overlays. *Journal of Network and Systems Management 13*, 2 (2005), 197–217. 5

[WBS02] Wald I., Benthin C., Slusallek P.: A simple and practical method for interactive ray tracing of dynamic scenes. *Submitted for publication, also available as a technical report at http://graphics. cs. uni-sb. de/Publications* (2002). 2

[WKB*02] Wald I., Kollig T., Benthin C., Keller A., Slusallek P.: Interactive global illumination using fast ray tracing. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 15–24. 2

[WRC88] Ward G. J., Rubinstein F. M., Clear R. D.: A ray tracing solution for diffuse interreflection. *ACM SIGGRAPH Computer Graphics 22*, 4 (1988), 85–92. 1

[WSBW01] Wald I., Slusallek P., Benthin C., Wagner M.: Interactive rendering with coherent ray tracing. In *Computer graphics forum* (2001), vol. 20, Wiley Online Library, pp. 153–165. 2

[ZKJ*01] Zhao B. Y., Kubiatowicz J., Joseph A. D., et al.: Tapestry: An infrastructure for fault-tolerant wide-area location and routing. 2